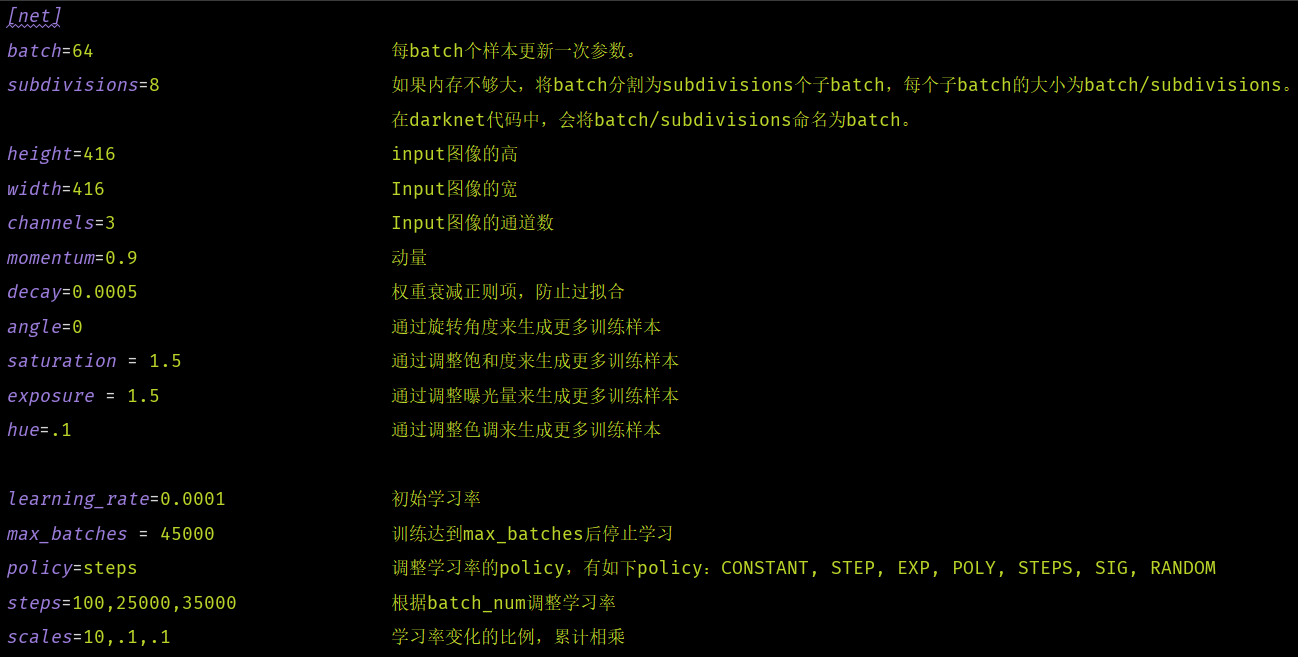
训练调参，这里以yolo网络模型举例（最近也在研究faster rcnn两阶段模型）

Net网络模块



batch：每次迭代要进行训练的图片数量

subdivision：batch中的图片再产生子集，源码中的图片数量int imgs = net.batch \* net.subdivisions \* ngpus

本例以上两个参数batch=64，subdivision=8，也就是说每轮迭代会从所有训练集里随机抽取 batch =64 个样本参与训练，所有这些 batch 个样本又被均分为 subdivision =8 次送入网络参与训练，以减轻内存占用的压力

width：输入图片宽度， height：输入图片高度，channels ：输入图片通道数

对于每次迭代训练，YOLO会基于角度(angle)，饱和度(saturation)，曝光(exposure)，色调(hue)产生新的训练图片

angle：图片角度变化，单位为度，假如 angle=5，就是生成新图片的时候随机旋转-5~5度

weight decay：权值衰减

防止过拟合，当网络逐渐过拟合时网络权值往往会变大，因此，为了避免过拟合，在每次迭代过程中以某个小因子降低每个权值，也等效于给误差函数添加一个惩罚项，常用的惩罚项是所有权重的平方乘以一个衰减常量之和。权值衰减惩罚项使得权值收敛到较小的绝对值。

saturation & exposure: 饱和度与曝光变化大小，tiny-yolo-voc.cfg中1到1.5倍，以及1/1.5~1倍

hue：色调变化范围，tiny-yolo-voc.cfg中-0.1~0.1

max\_batches：最大迭代次数

learning rate：学习率

学习率决定了参数移动到最优值的速度快慢，如果学习率过大，很可能会越过最优值导致函数无法收敛，甚至发散；反之，如果学习率过小，优化的效率可能过低，算法长时间无法收敛，也易使算法陷入局部最优（非凸函数不能保证达到全局最优）。合适的学习率应该是在保证收敛的前提下，能尽快收敛。

设置较好的learning rate，需要不断尝试。在一开始的时候，可以将其设大一点，这样可以使weights快一点发生改变，在迭代一定的epochs之后人工减小学习率。

policy：调整学习率的策略

调整学习率的policy，有如下policy：CONSTANT, STEP, EXP, POLY，STEPS, SIG, RANDOM

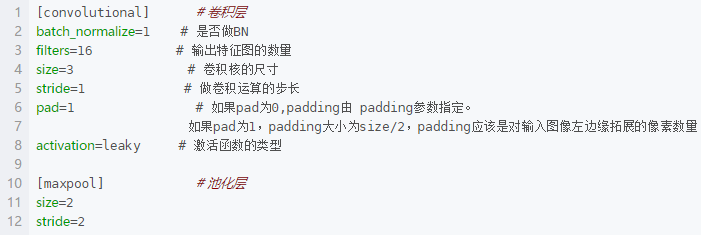
steps：学习率变化时的迭代次数

根据batch\_num调整学习率，若steps=100,20000,30000，则在迭代100次，20000次，30000次时学习率发生变化，该参数与policy中的steps对应

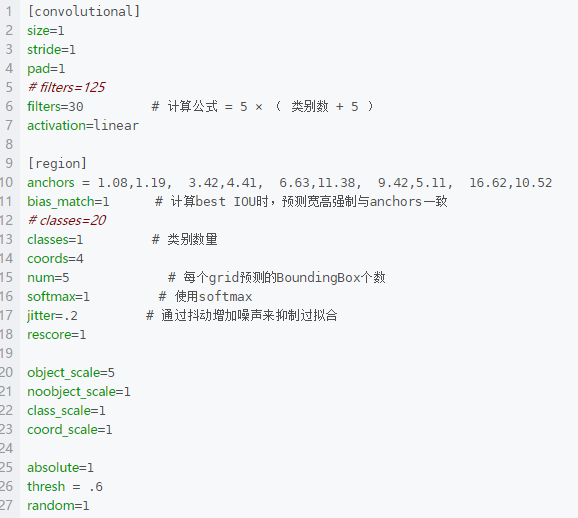
scales：学习率变化的比率

相对于当前学习率的变化比率，累计相乘，与steps中的参数个数保持一致

中间部分



结束部分



anchors

预测框的初始宽高，第一个是w，第二个是h，总数量是num\*2，YOLOv2作者说anchors是使用K-MEANS获得，其实就是计算出哪种类型的框比较多，可以增加收敛速度，如果不设置anchors，默认是0.5，还有就是anchors读入参数中名字是biases

bias\_match

如果为1，计算best iou时，预测宽高强制与anchors一致

coords：BoundingBox的tx,ty,tw,th

tx与ty是相对于左上角的gird，同时是当前grid的比例，tw与th是宽度与高度取对数

jitter：通过抖动增加噪声来抑制过拟合

YOLOv2中使用的是crop，filp，以及net层的angle，flip是随机的，crop就是jitter的参数，tiny-yolo-voc.cfg中jitter=.2，就是在0~0.2中进行crop

rescore：决定使用哪种方式计算IOU的误差

为1时，使用当前best iou计算，为0时，使用1计算

object\_scale & noobject\_scale & class\_scale & coord\_scale

YOLOv1论文中cost function的权重，哪一个更大，每一次更新权重的时候，对应方面的权重更新相对比重更大

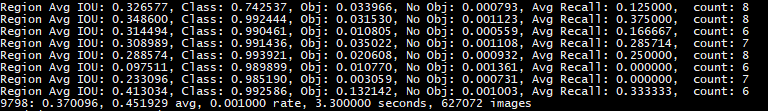
thresh：决定是否需要计算IOU误差的参数

大于thresh，IOU误差不会夹在cost function中

random

random为1时会启用Multi-Scale Training，随机使用不同尺寸的图片进行训练,如果为0，每次训练大小与输入大小一致;

以下是训练过程中终端输出的一个[截图](https://www.baidu.com/s?wd=%E6%88%AA%E5%9B%BE&tn=24004469_oem_dg&rsv_dl=gh_pl_sl_csd" \t "_blank)：

  
以上[截图](https://www.baidu.com/s?wd=%E6%88%AA%E5%9B%BE&tn=24004469_oem_dg&rsv_dl=gh_pl_sl_csd" \t "_blank)显示了所有训练图片的一个批次（batch），批次大小的划分根据我们在 .cfg 文件中设置的subdivisions参数。在我使用的 .cfg 文件中 batch = 64 ，subdivision = 8，所以在训练输出中，训练迭代包含了8组，每组又包含了8张图片，跟设定的batch和subdivision的值一致。（注： 也就是说每轮迭代会从所有训练集里随机抽取 batch = 64 个样本参与训练，所有这些 batch 个样本又被均分为 subdivision = 8 次送入网络参与训练，以减轻内存占用的压力）

针对上图中最后一行中的信息，我们来一步一步的分析。如下的输出是由 detector.c 生成的https://img-blog.csdn.net/20171117213650195

* 9798： 指示当前训练的迭代次数
* 0.370096： 是总体的Loss(损失）
* 0.451929 avg： 是平均Loss，这个数值应该越低越好，一般来说，一旦这个数值低于0.060730 avg就可以终止训练了。
* 0.001000 rate： 代表当前的学习率，是在.cfg文件中定义的。
* 3.300000 seconds： 表示当前批次训练花费的总时间。
* 627072 images： 这一行最后的这个数值是9798\*64的大小，表示到目前为止，参与训练的图片的总量。

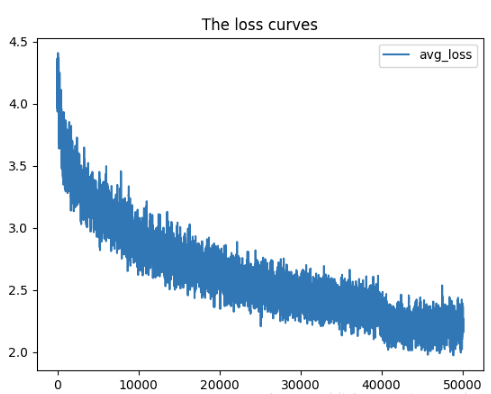
我们来分析一下这些用来描述训练图集中的一个批次的训练结果的输出。

https://img-blog.csdn.net/20171117214147246

* Region Avg IOU: 0.326577： 表示在当前subdivision内的图片的平均IOU，代表预测的矩形框和真实目标的交集与并集之比，这里是32.66%，这个模型需要进一步的训练。
* Class: 0.742537： 标注物体分类的正确率，期望该值趋近于1。
* Obj: 0.033966： 越接近1越好。
* No Obj: 0.000793： 期望该值越来越小，但不为零。
* Avg Recall: 0.12500： 是在recall/count中定义的，是当前模型在所有subdivision图片中检测出的正样本与实际的正样本的比值。在本例中，只有八分之一的正样本被正确的检测到。
* count: 8：count后的值是所有的当前subdivision图片（本例中一共8张）中包含正样本的图片的数量。在输出log中的其他行中，可以看到其他subdivision也有的只含有6或7个正样本，说明在subdivision中含有不含检测对象的图片。

举例如下：

* 训练时长：
* loss：最后稳定在2.0-2.5之间
* AP：0.45655
* loss曲线：如下图：



分析及总结：

可以看到avg loss在35000个batch的训练后，几乎不再下降，但是当到40000个batch后，学习率变为原来的0.1之后，loss又进一步下降，但是当45000个batch以后有些许上升的趋势，一方面是因为学习率可能难以下降，另一方面可能有学习率在45000个batch之后进一步变为之前的0.1。

最终的结果中显示对于以下几类小物体识别效果很差：

bird : 0.297

boat : 0.281

bottle : 0.172

potted plant : 0.227

改进：

思路1：改变学习率step模式，在45000个bacth后继续使用原学习率，验证loss曲线变化;

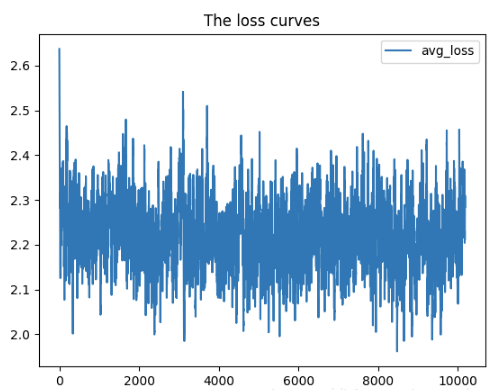
思路2：改变网络结构，增加小尺度anchor，将原来是yolov3-tiny中的两个yolo层改为3个，anchor增加为9个。

改进后实验结果：

改变学习率step模式：结果AP=0.45715，有微小幅度的上涨，至少证明45000个batchuu以后的学习率调整为0.1是无意义的，或者说过于小。

在47000个bacth以后loss几乎无下降。

loss curve（40000个batch以后）:



发现在大约48000-50000的时候损失达到最小，于是又用50000次batch时备份的weight 文件验证一下，发现，AP竟然等于0.47185！，可以说是效果有较大的改善，鬼知道，从50000次到50200次这200次batch的训练发生了什么，竟然使效果发生那么大的下降。

改变网络结构增加小尺度anchor：

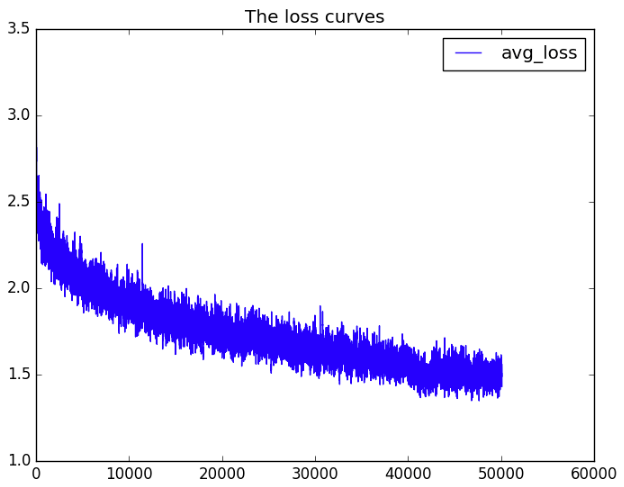
更改：在第一次训练的配置文件的基础上，将多尺度预测里的尺度因子，增加一个尺度，变为三个，看会不会效果更好一下，于是更改网络结构，将21层卷积层的输出经过上采样，与第6层的52\*52的feature map合并，增加一层52\*52的feature map进行预测。anchors按照yolov3一样进行配置，共9 个，每个尺度3个anchor。其他训练参数保持不变。

训练时长：因为多加了一个yolo层，整体训练时间有较大的增加，另外显存爆出了out of memory 的错误，所以，将subdivision翻倍，综合造成的影响是，每个batch的训练时间大约增加了1.5倍-2倍。

loss：目前在迭代batch次数较少的情况下，已经小于第一次训练了。

AP：0.44045（显著下降）

loss curve：



这就使我很困惑了，loss曲线都比第一次要好看非常多，但是为什么效果反而变差了。这就不知道该怎么解释了。